A Software Design for Solving Coupled Multiphysics Equations in a Dynamic, Extensible and Efficient Way

P. K. Notz\textsuperscript{1}, R. P. Pawlowski\textsuperscript{2} and J. Sutherland\textsuperscript{3}

\textsuperscript{1} Engineering Sciences Center, Sandia National Laboratories\textsuperscript{4}, PO Box 5800, MS 0836, Albuquerque, NM 87185-0836, USA, pknotz@sandia.gov
\textsuperscript{2} Computation, Computers and Math, Sandia National Laboratories\textsuperscript{4}, PO Box 5800, MS 1318, Albuquerque, NM 87185-1318, USA, rppawlo@sandia.gov
\textsuperscript{3} University of Utah, 50 South Central Campus Drive, Room 3250, Salt Lake City, Utah 84112-1114, USA, James.Sutherland@utah.edu
\textsuperscript{4} Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy’s National Nuclear Security Administration under contract DE-AC04-94AL85000.

Recent advances in hardware and software computational technology greatly increase the space of problems that are amenable to modeling and simulation. Parallel architectures, both distributed and multicore, provide resources that permit the solution of problems with ever increasing resolution and complexity. Multiphysics simulations typically include several coupled PDEs and involve a vast selection of material models, constitutive models, source terms, PDE formulations and discretizations. Additionally, multiscale, multiphysics simulations rely on hierarchical models, which typically have limited ranges of validity. As a simulation realizes different physical regimes, different models may be appropriate. However, dynamically transitioning from one model to another may require significant and intrusive changes such as: addition and/or removal of transport equations, modification of constitutive relationships, modification/addition/removal of source terms, modification of numerical algorithms, the reversal of dependent and independent variables, etc.

Demands for increasing model complexity are reflected by an increasing complexity in the software. Likewise, the growth in sophistication of the software is reflected by a growth in the cost of development and maintenance. In order to leverage these significant investments, the software must be open to the addition of new physics, support the use of different discretizations, adapt to changes in formulations and accommodate couplings that were not anticipated at the time of implementation. New strategies and software designs are required to meet these growing needs in a robust, efficient and cost-effective way.

Many large scale scientific computing codes use a computational framework that provides parallel IO, data management, and MPI parallelization services. Examples of such frameworks include SIERRA\textsuperscript{1,2}, Uintah\textsuperscript{3}, and SAMRAI\textsuperscript{4,5}. By abstracting data management and parallelization these frameworks enable application programmers to focus on algorithm implementation. What’s most lacking currently is a set of abstractions that support the aforementioned requirements of modern multiphysics software and simultaneously minimize the accidental complexity in algorithm implementation. In this talk, we describe a new design approach\textsuperscript{6} to address this need.

The primary source of algorithmic complexity in mechanics software is due to a focus on the flow of data rather than data dependencies. In this paper, we present a software design that exposes dependencies among operations on data and thus allows for automated organization of the operation sequence. This design paradigm requires programmers to explicitly declare data dependencies and, in turn, removes the need for them to understand the complex interdependencies inherently present in multiphysics software. In this approach, complex problems are decomposed into a directed acyclic graph of data where edges represent dependencies. From this graph, a wealth of information can be derived using standard graph algorithms. We describe the design and demonstrate its effectiveness at enabling the support of rapid
development, separation of concerns, extensibility and multicore parallelization. Ancillary benefits of the proposed design include improved testability, reduced defect rates, increased code re-use and simpler code that is easier to both read and write.

As with any design, compromises and trade-offs are made with the proposed approach. The benefits of this design have consequences to memory utilization, compile-time optimization, runtime performance and understandability of the software. As part of this talk, we describe our experiences with this approach in the context of these trade-offs.

An open source implementation of this design is available in the Phalanx⁷ package of the Trilinos⁸ software suite.

References


